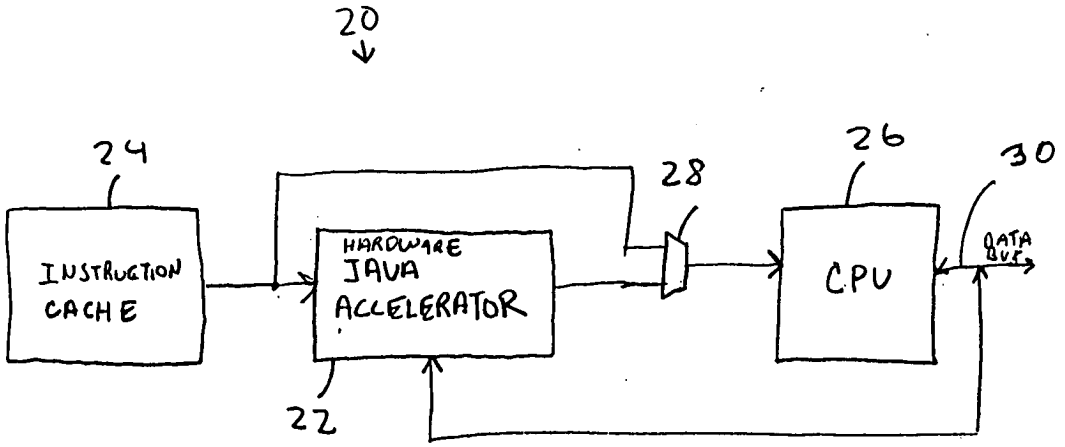

FIGURE 1

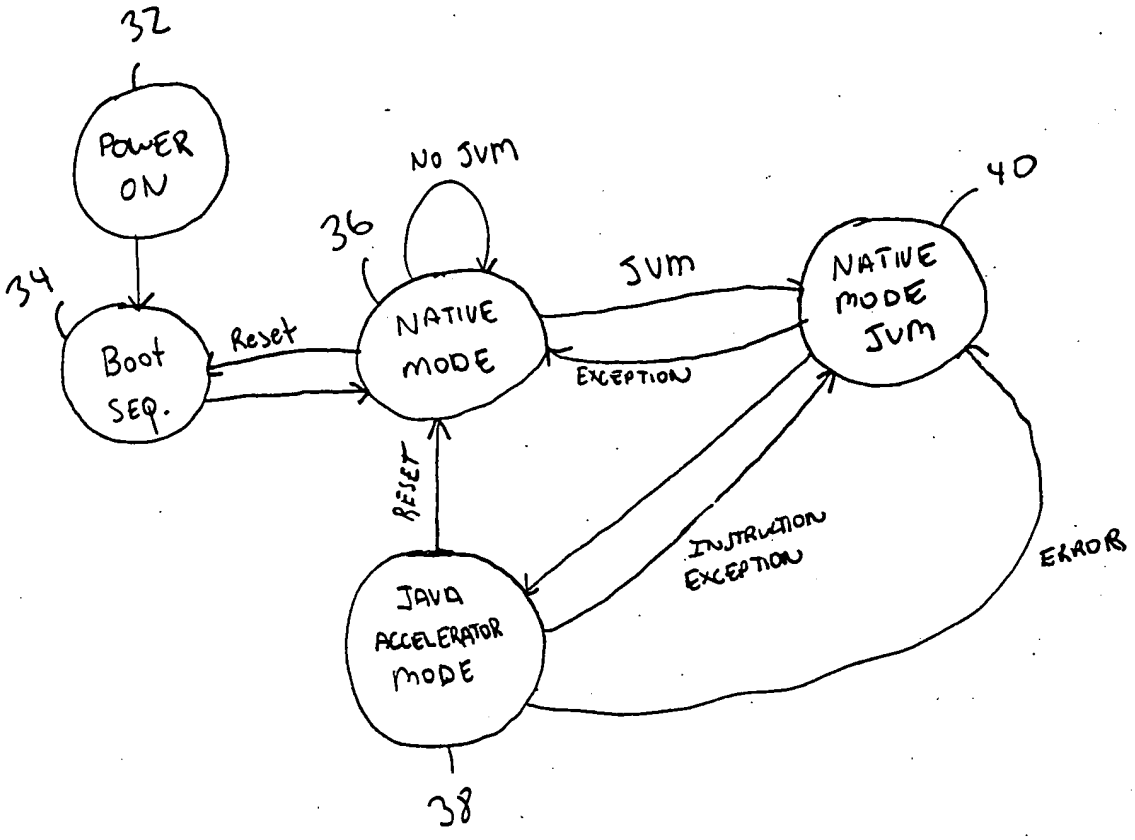

FIGURE 2

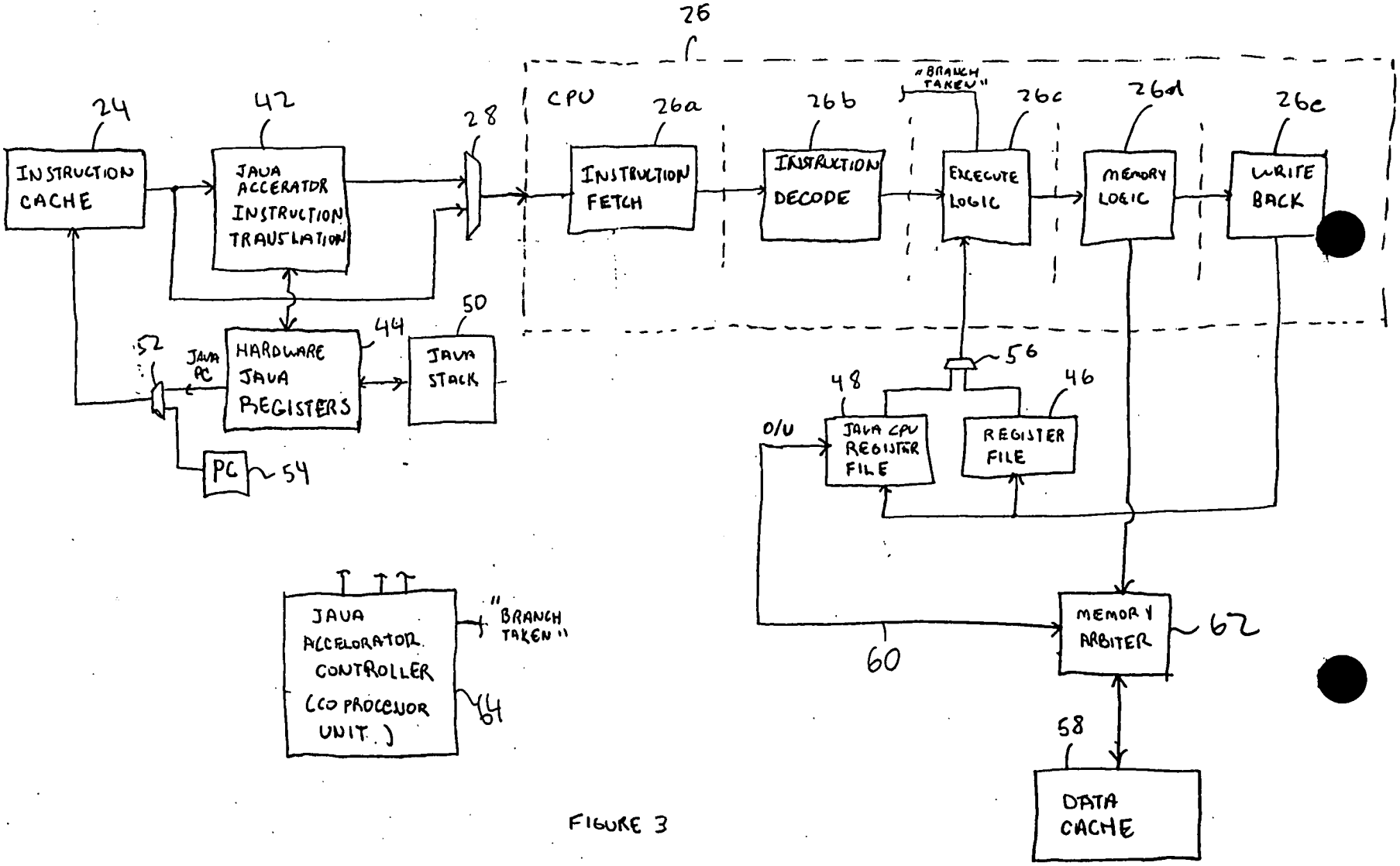
26
CPU
24
INSTRUCTION CACHE
42
JAVA ACCERATOR INSTRUCTION TRANSLATION
28
26a
INSTRUCTION FETCH
26b
INSTRUCTION DECODE
"BRANCH TAKEN"
26c
EXECUTE LOGIC
26d
MEMORY LOGIC
26e
WRITE BACK
50
44
52
JAVA PC
HARDWARE JAVA REGISTERS
JAVA STACK
PC
54
56
48
46
O/U
JAVA CPU REGISTER FILE
REGISTER FILE
JAVA ACCELERATOR CONTROLLER (CO PROCESSOR UNIT)
64
"BRANCH TAKEN"
60
MEMORY ARBITER
62
58
DATA CACHE

FIGURE 3

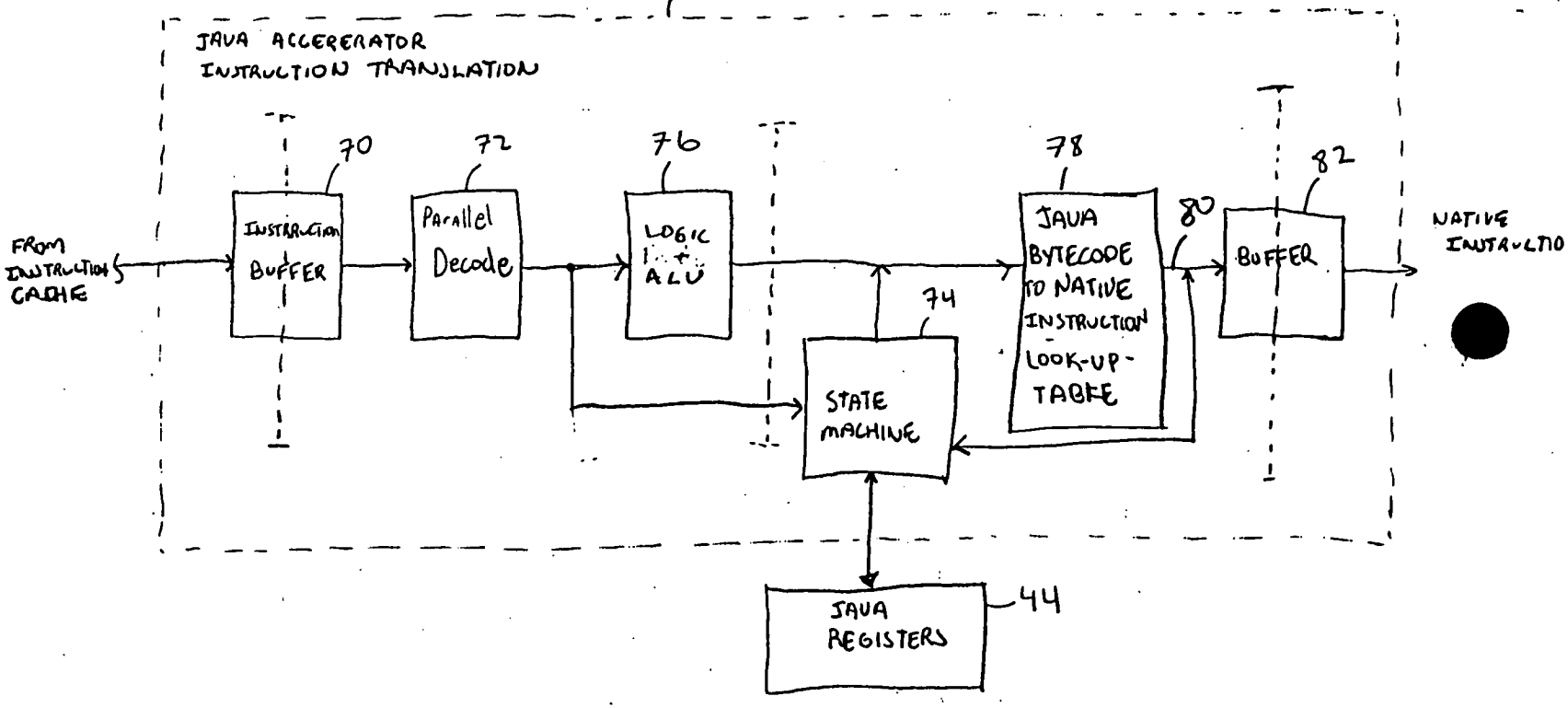
42
JAVA ACCELERATOR
INSTRUCTION TRANSLATION
FROM INSTRUCTION CACHE
INSTRUCTION BUFFER
70
PARALLEL DECODE
72
LOGIC & ALU
76
STATE MACHINE
74
JAVA BYTECODE TO NATIVE INSTRUCTION - LOOK-UP - TABLE
78
80
BUFFER
82
NATIVE INSTRUCTIO
JAVA REGISTERS
44

FIGURE 4

I. INSTRUCTION TRANSLATION

JAVA BYTECODE	⇒	NATIVE INSTRUCTION
iadd		ADD R1, R2

II. JAVA REGISTER

PC = VALUE A	⇒	PC = VALUE A + 1
OPTOP = VALUE B (R1)		OPTOP = VALUE B - 1 (R2)
VAR = VALUE C		VAR = VALUE C

III. JAVA CPU REGISTER FILE

	R0	0001		R0	0001
contains VALUE → top of operand stack	R1	0150	Not a valid stack value →	R1	0150
	R2	1210	contains →	R2	1360
	R3	0007	value of the top of operand stack	R3	0007
	R4	0005		R4	0005
	R5	0006		R5	0006
contains → first VARIABLE	R6	1221		R6	1221
	R7	1361		R7	1361

IV. MEMORY

OPTOP =	VALUE B ⇒	0150			0150
	(VALUE B-1) -	1210	OPTOP = VALUE B-1	-	1360
	-	0007		-	0007
	-	0005		-	0005
	-	0006		-	0006
	-	0001		-	0001
	-	4427		-	4427
VAR =	VALUE C -	1221	VAR = VALUE C	-	1221
	-	1361		-	1361
	-	1101		-	1101

FIGURE 5

I. INSTRUCTION TRANSLATION

JAVA BYTECODE		NATIVE INSTRUCTION
iload_n iadd	⇒	ADD R6, R1

II. JAVA REGISTER

Before		After
PC = VALUE A		PC = VALUE A + 2
OPTOP = VALUE B (R1)	⇒	OPTOP = VALUE B (R1)
VAR = VALUE C		VAR = VALUE C

III. JAVA CPU REGISTER FILE

	Register	Value	⇒		Register	Value
	R0	0001			R0	0001
contains value of top of operand stack →	R1	0150		contains value of top of stack →	R1	1371
	R2	1210			R2	1210
	R3	0007			R3	0007
	R4	0005			R4	0005
	R5	0006			R5	0006
CONTAIN FIRST VARIABLE →	R6	1221		contain first variable →	R6	1221
	R7	1361			R7	1361

IV. MEMORY

	Before		After
OPTOP = VALUE B	0150	OPTOP = VALUE B	1371
	1210		1210
	0007		0007
	0005		0005
	0006		0006
	0001		0001
	4427		4427
~~		~~~	
VAR = VALUE C	1221	VAR = VALUE C	1221
	1361		1361
	1101		1101

FIGURE 6

Opcodes Mnemonic	Opcode xHH	Excep Gen
nop	0x00	
aconst_null	x01	
iconst_m1	x02	
iconst_n(0-5)	x03 - x08	
lconst_n(0-1)	x09 - x0a	
fconst_n(0-2)	x0c - x0d	
dconst_n(0-1)	x0e -x0f	
bipush	x10	
sipush	x11	
ldc	x12	y
ldc_w	x13	y
ldc2_w	x14	y
iload	x15	
lload	x16	
fload	x17	
dload	x18	
aload	x19	
iload_n(0-3)	x1a - x1d	
lload_n(0-3)	x1e - x21	
fload_n(0-3)	x22 - x25	
dload_n(0-3)	x26 - x29	
aload_n(0-3)	x2a - x2d	
iaload	x2e	
laload	x2f	
faload	x30	
daload	x31	
aaload	x32	
baload	x33	
caload	x34	
saload	x35	
istore	x36	
lstore	x37	
fstore	x38	
dstroe	x39	
astroe	x3a	
istore_n(0-3)	x3b - x3e	
lstore_n(0-3)	x3f - x42	
fstore_n(0-3)	x43 - x46	
dstore_n(0-3)	x47 - x4a	
astore_n(0-3)	x4b - x4e	
iastore	x4f	
lastore	x50	
fastroe	x51	
dastore	x52	
bastore	x53	
aastore	x54	
castroe	x55	
sastore	x56	

FIGURE 7A

pop	x57	
pop2	x58	
dup	x59	
dup_x1	x5a	
dup_x2	x5b	
dup2	x5c	
dup2_x1	x5d	
dup2_x2	x5e	
swap	x5f	
iadd	x60	
ladd	x61	
fadd	x62	y
dadd	x63	y
isub	x64	
lsub	x65	
fsub	x66	y
dsub	x67	y
imul	x68	
lmul	x69	
fmul	x6a	y
dmul	x6b	y
idiv	x6c	y
ldiv	x6d	y
fdiv	x6e	y
ddiv	x6f	y
irem	x70	y
lrem	x71	y
frem	x72	y
drem	x73	y
ineg	x74	
lneg	x75	
fneg	x76	y
dneg	x77	y
ishl	x78	
lshl	x79	
ishr	x7a	
lshr	x7b	
iushr	x7c	
lushr	x7d	
iand	x7e	
land	x7f	
ior	x80	
lor	x81	
ixor	x82	
lxor	x83	
iinc	x84	
i2l	x85	y
i2f	x86	y
i2d	x87	y
l2i	x88	y
l2f	x89	y
l2d	x8a	y

FIGURE 7B

f2i	x8b	y
f2l	x8c	y
f2d	x8d	y
d2i	x8e	y
d2l	x8f	y
d2f	x90	y
i2b	x91	
i2c	x92	
i2s	x93	
lcmp	x94	y
fcmpl	x95	y
fcmpg	x96	y
dcmpl	x97	y
dcmpg	x98	y
ifeq	x99	
ifne	x9a	
iflt	x9b	
ifge	x9c	
ifgt	x9d	
ifle	x9e	
if_icmpeq	x9f	
if_icmpne	xa0	
if_icmplt	xa1	
if_acmpge	xa2	
if_cmpgt	xa3	
if_icmple	xa4	
if_acmpeq	xa5	
if_acmpne	xa6	
goto	xa7	
jsr	xa8	
ret	xa9	
tableswitch	xaa	y
lookupswitch	xab	y
ireturn	xac	
lreturn	xad	
freturn	xae	
dreturn	xaf	
areturn	xb0	
return	xb1	
getstatic	xb2	y
putstatic	xb3	y
getfield	xb4	y
putfield	xb5	y
invokevirtual	xb6	y
invokespecial	xb7	y
invokestatic	xb8	y
invokeinterface	xb9	y
xxunsedxxx	xba	y
new	xbb	y
newarray	xbc	y
anewarray	xbd	y
arraylength	xbe	y

FIGURE 7C

athrow	xbf	y
checkcast	[illegible]o	y
instanceof	[illegible]c1	y
monitorenter	xc2	y
monitorexit	xc3	y
wide	xc4	y
multianewarray	xc5	y
ifnull	xc6	y
ifnonnull	xc7	y
goto_w	xc8	
jsr_w	xc9	
ldc_quick	xcb	y
ldc_w_quick	xcc	y
ldc2_w_quick	xcd	y
getfield_quick	xce	y
putfield_quick	xcf	y
getfield2_quick	xd0	y
putfield2_quick	xd1	y
getstatic_quick	xd2	y
putstatic_quick	xd3	y
gtestatic2_quick	xd4	y
putstatic2_quick	xd5	y
invokevirtual_quick	xd6	y
invokenonvirtual_quick	xd7	y
invokesuper_quick	xd8	y
invokestatic_quick	xd9	y
invokeinterface_quick	xda	y
invokevirtualobject_quick	xdb	y
new_quick	xdc	y
anewarray_quick	xde	y
multinewarray_quick	xdf	y
checkcast_quick	xe0	y
instanceof_quick	xe1	y
invokevirtual_quick_w	xe2	y
getfield_quick_w	xe3	y
putfield_quick_w	xe4	y
breakpoint	xca	y
impdep1	xfe	y
impdep2	xff	y

FIGURE 7 D

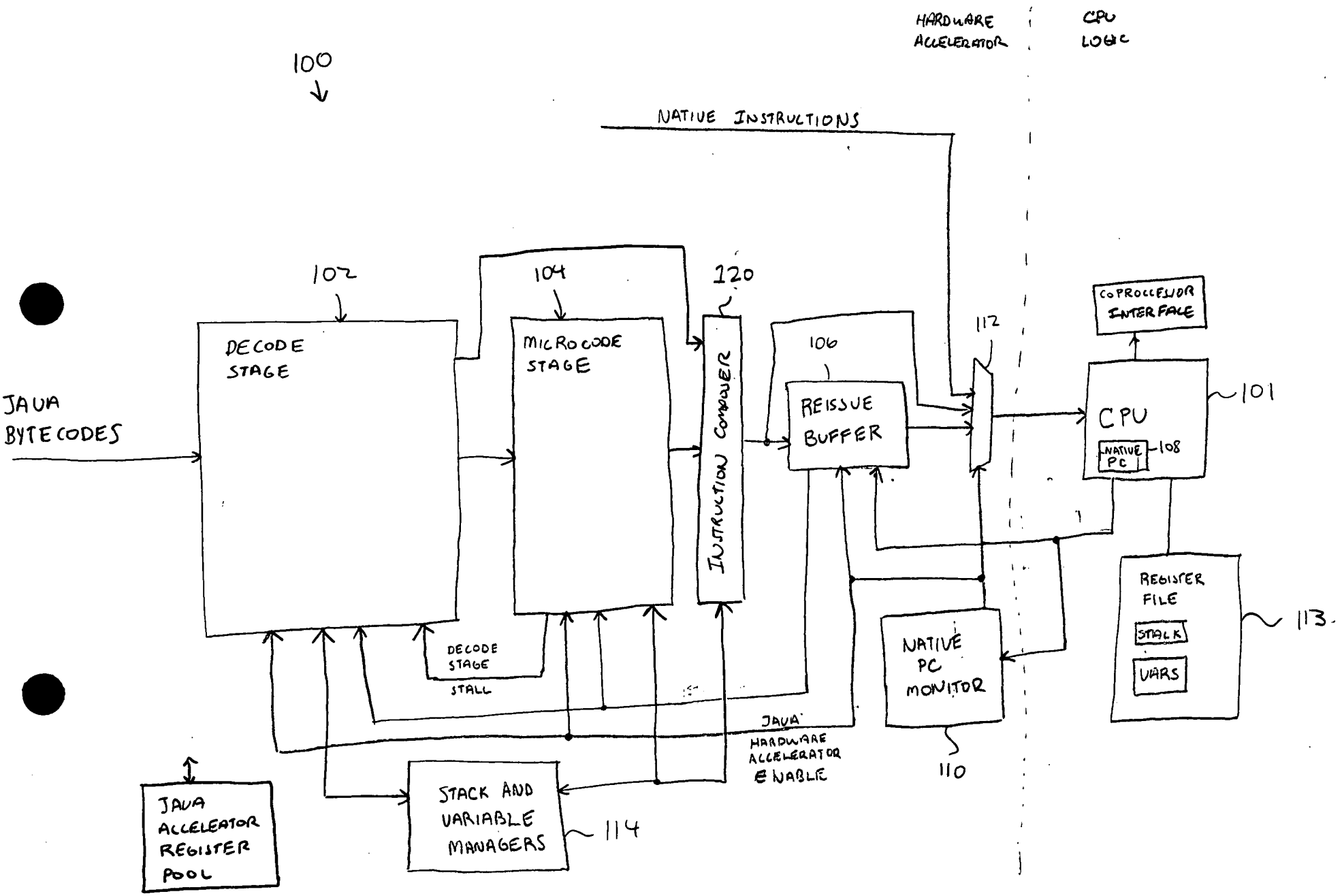

FIGURE 8

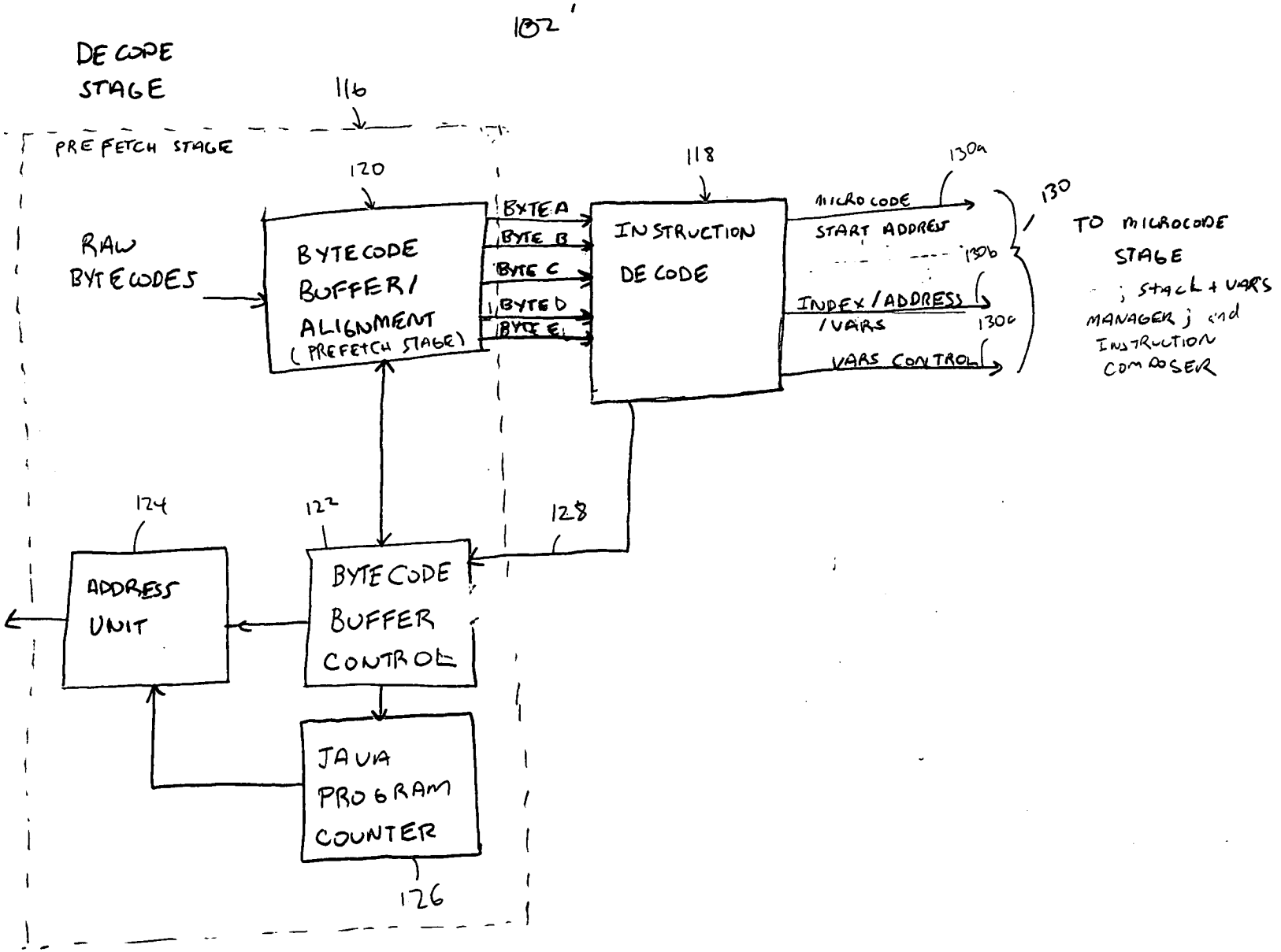

FIGURE 1

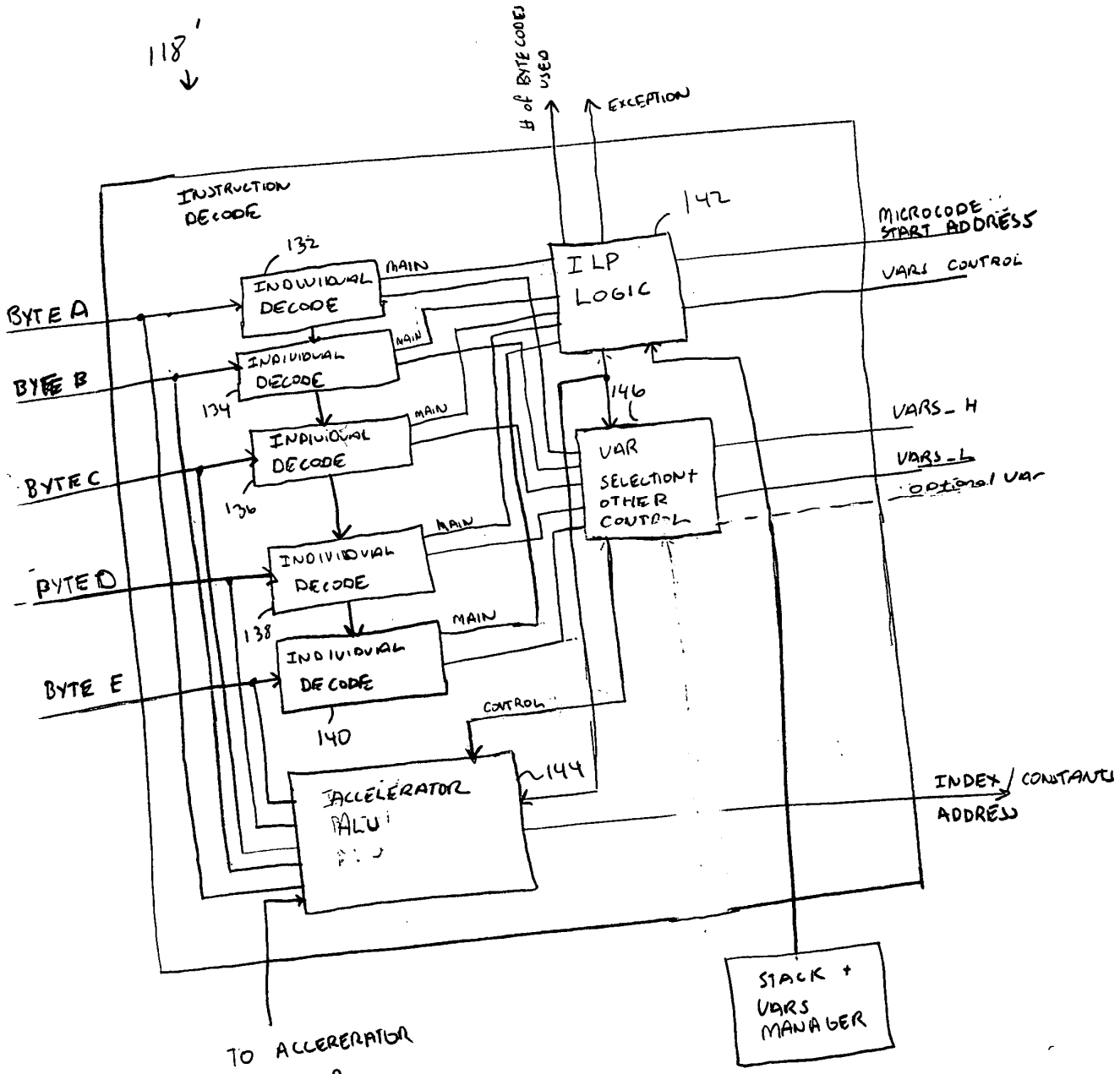
118'
# of BYTE CODES USED
EXCEPTION
INSTRUCTION DECODE
132
INDIVIDUAL DECODE
MAIN
142
ILP LOGIC
MICROCODE START ADDRESS
VARS CONTROL
BYTE A
BYTE B
134
INDIVIDUAL DECODE
MAIN
146
BYTE C
INDIVIDUAL DECODE
136
VARS_H
VAR SELECTION OTHER CONTROL
VARS_L
optional VAR
MAIN
BYTE D
INDIVIDUAL DECODE
138
MAIN
BYTE E
INDIVIDUAL DECODE
140
CONTROL
144
ACCELERATOR ALU
INDEX / CONTANTS ADDRESS
STACK + VARS MANAGER
TO ACCELERATOR REGISTER

FIGURE 10

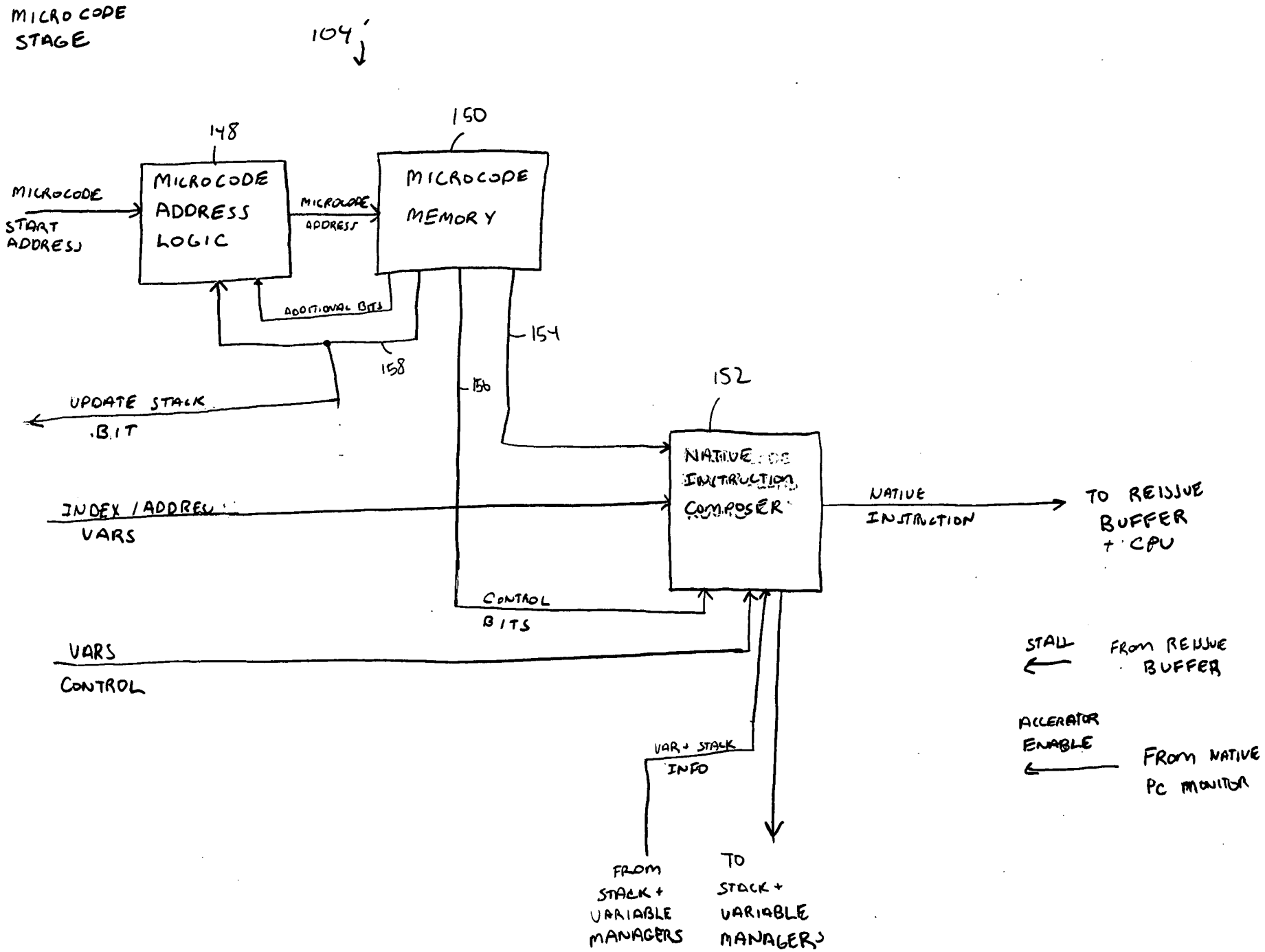

Figure 17

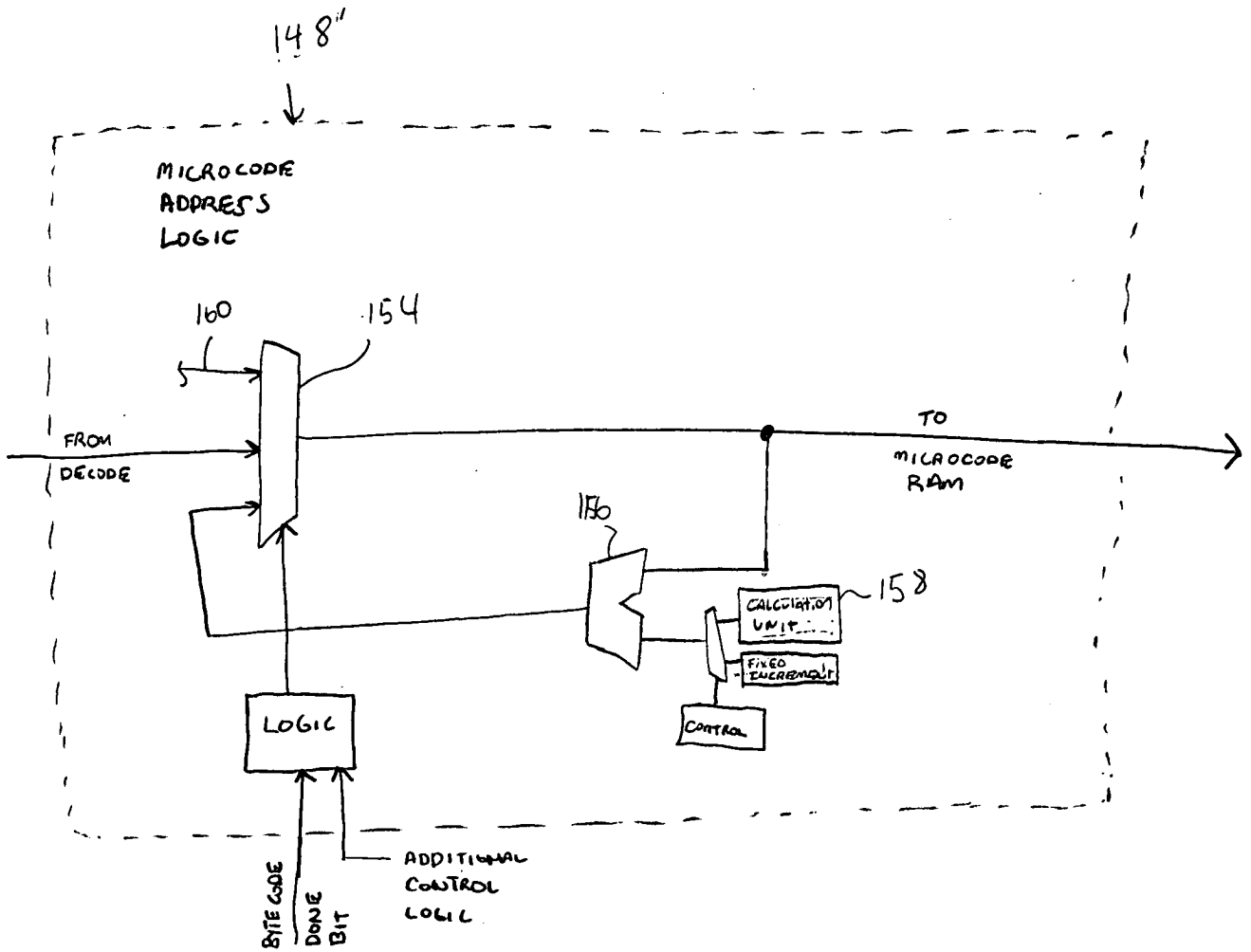

FIGURE 12

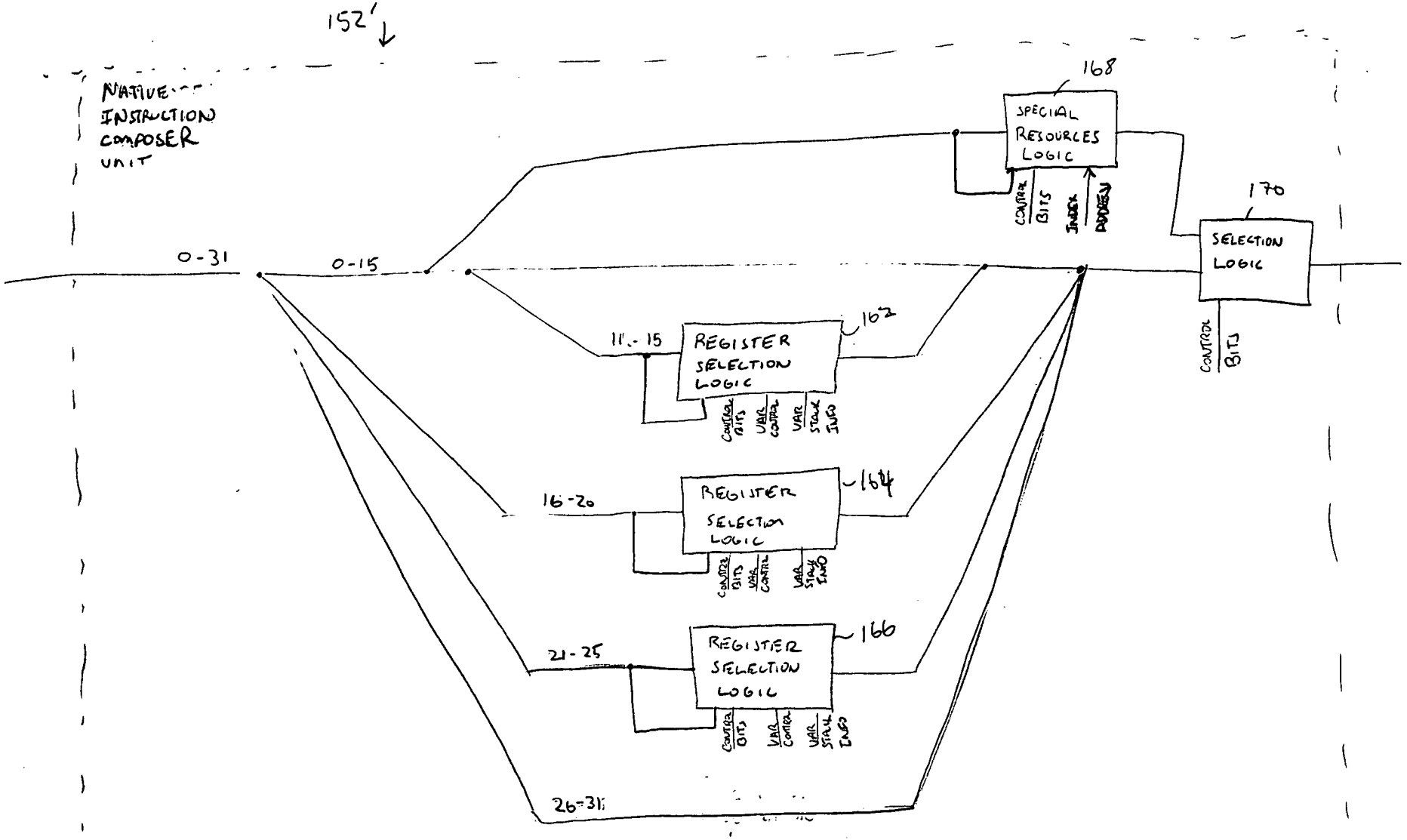

FIGURE 13

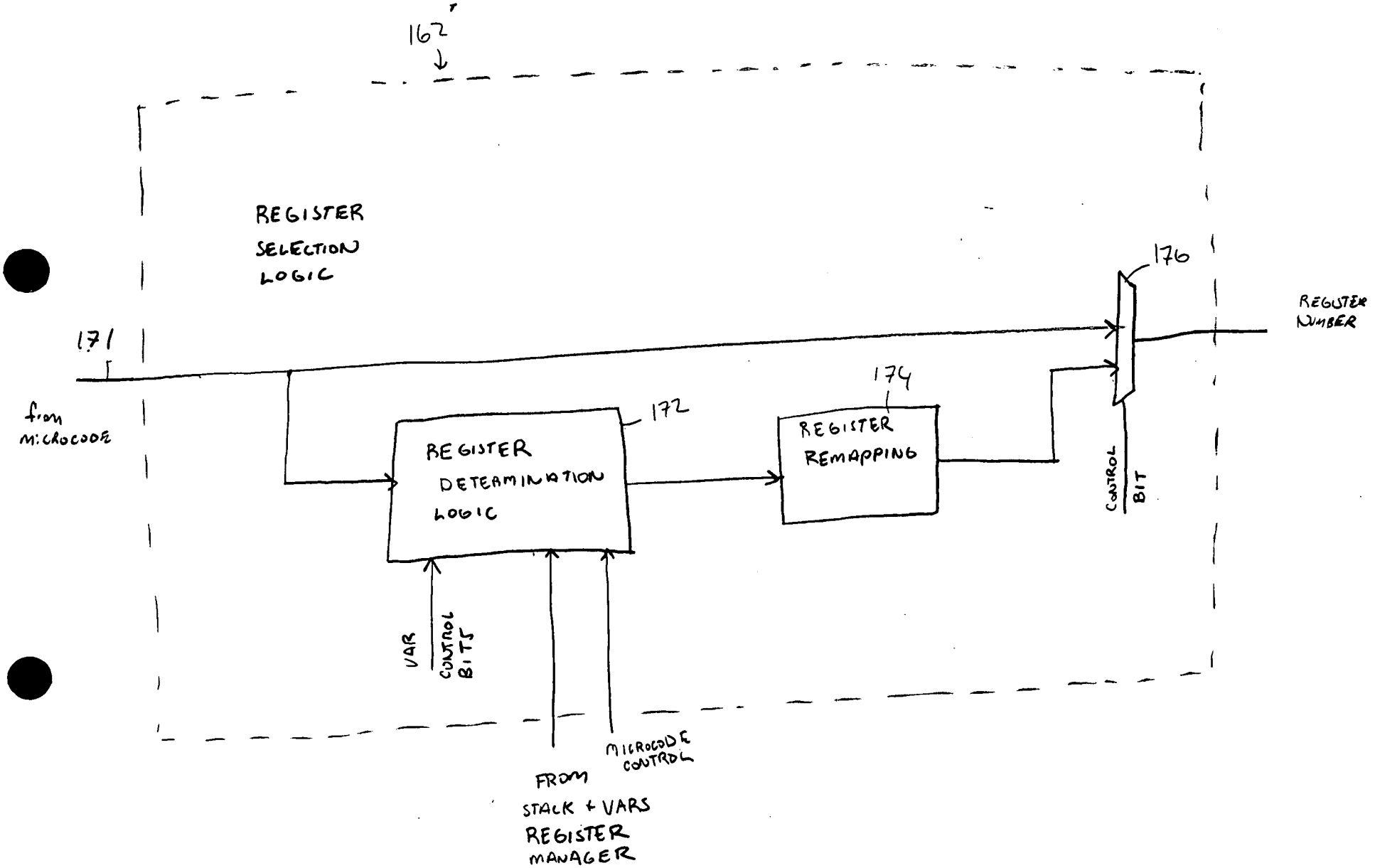

FIGURE 14

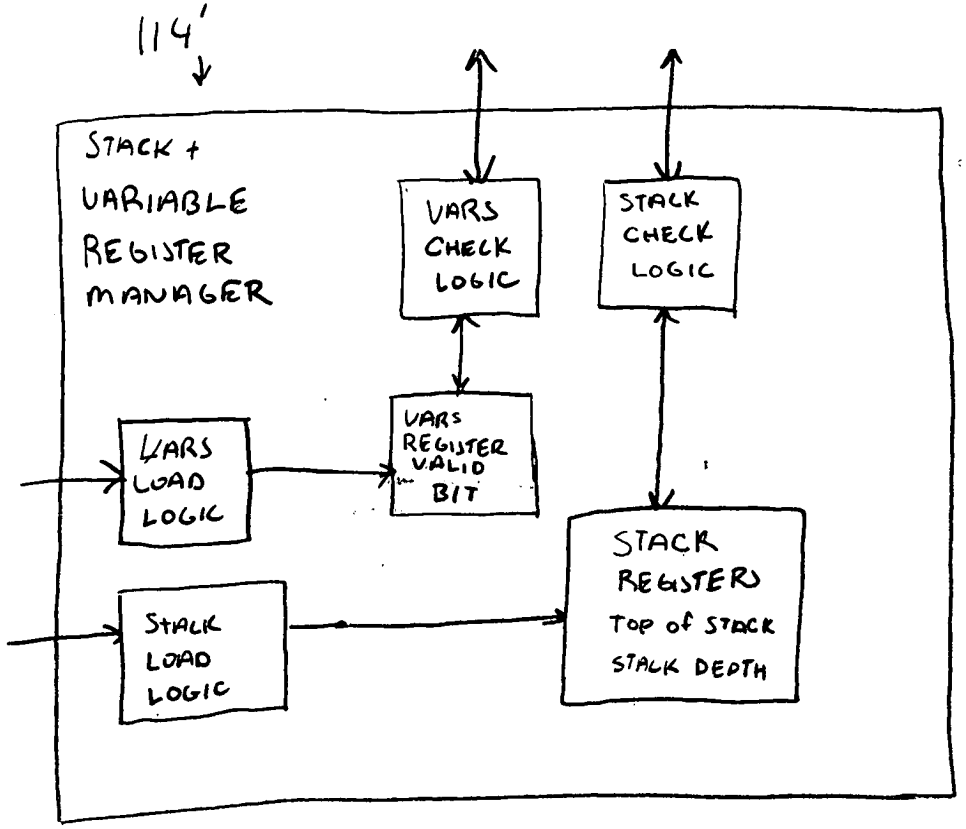
114'
STACK + VARIABLE REGISTER MANAGER
VARS CHECK LOGIC
STACK CHECK LOGIC
VARS REGISTER VALID BIT
VARS LOAD LOGIC
STACK LOAD LOGIC
STACK REGISTERS
TOP of STACK
STACK DEPTH

FIGURE 15

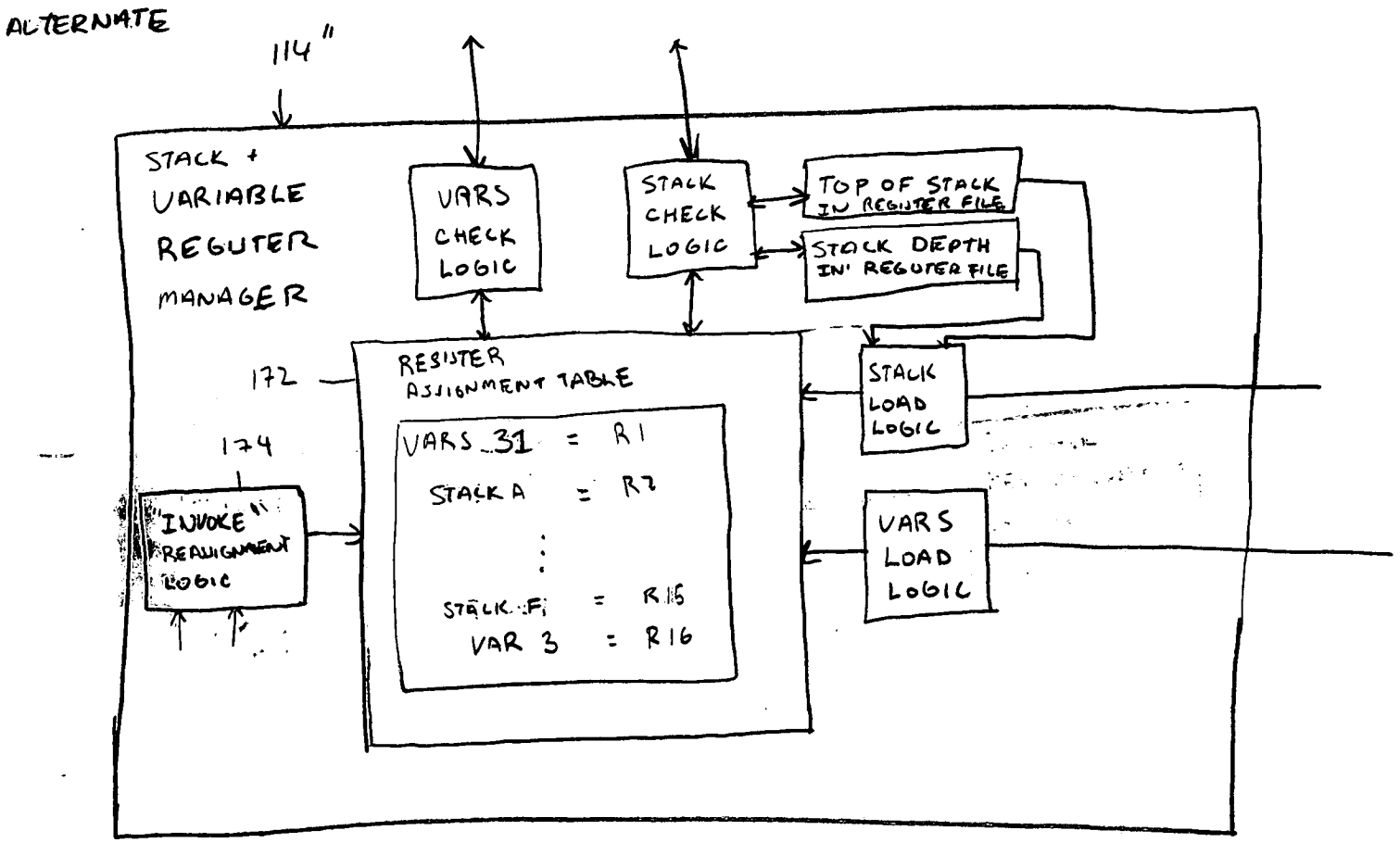
ALTERNATE
114"
STACK + VARIABLE REGISTER MANAGER
VARS CHECK LOGIC
STACK CHECK LOGIC
TOP OF STACK IN REGISTER FILE
STACK DEPTH IN REGISTER FILE
172
REGISTER ASSIGNMENT TABLE
174
VARS_31 = R1
STACK A = R2
STACK F = R15
VAR 3 = R16
INVOKE REALIGNMENT LOGIC
STACK LOAD LOGIC
VARS LOAD LOGIC

FIGURE 16

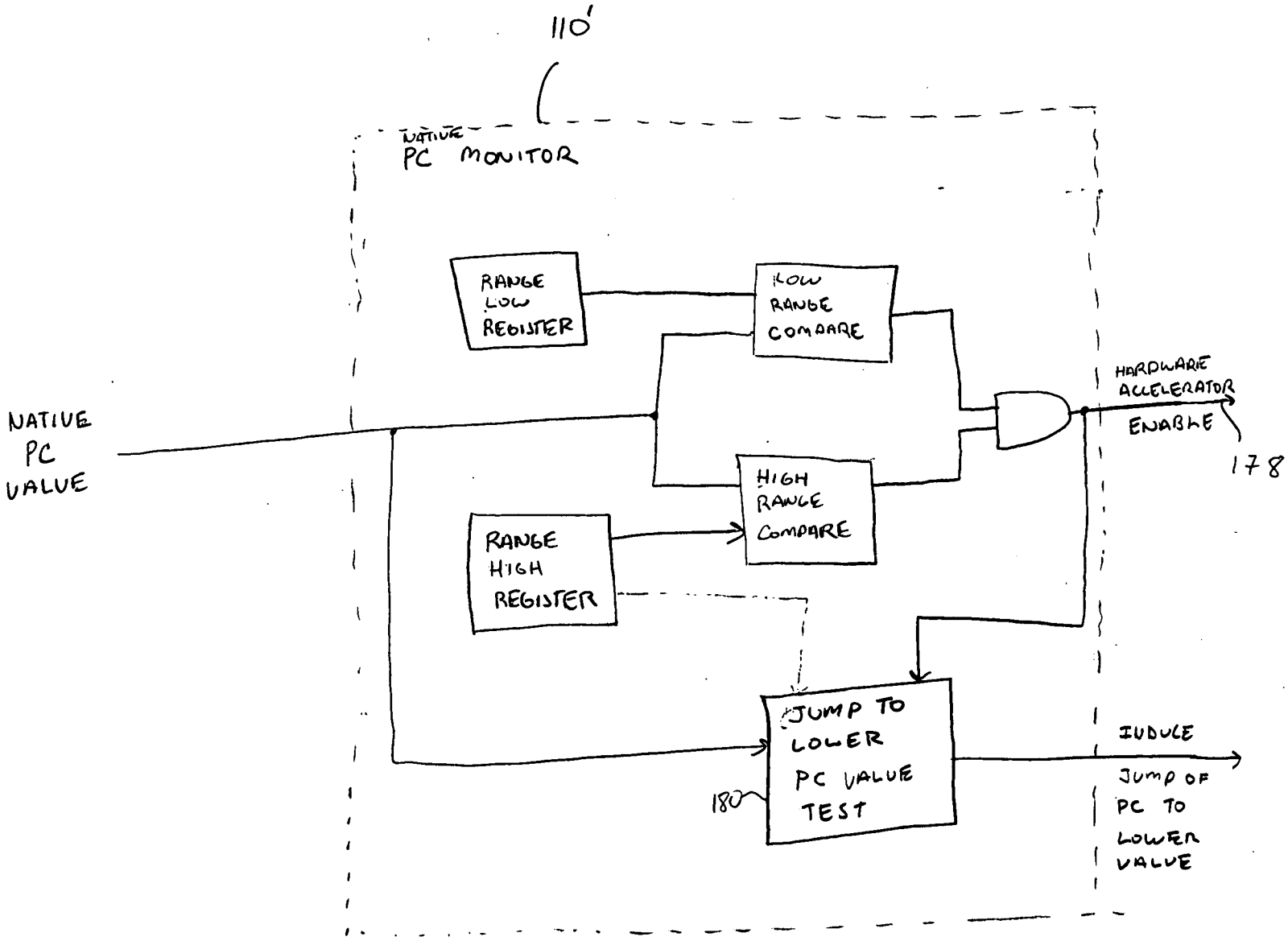
110'
NATIVE
PC MONITOR
RANGE
LOW
REGISTER
LOW
RANGE
COMPARE
HIGH
RANGE
COMPARE
RANGE
HIGH
REGISTER
NATIVE
PC
VALUE
HARDWARE
ACCELERATOR
ENABLE
178
JUMP TO
LOWER
PC VALUE
TEST
180
INDUCE
JUMP OF
PC TO
LOWER
VALUE

FIGURE 17

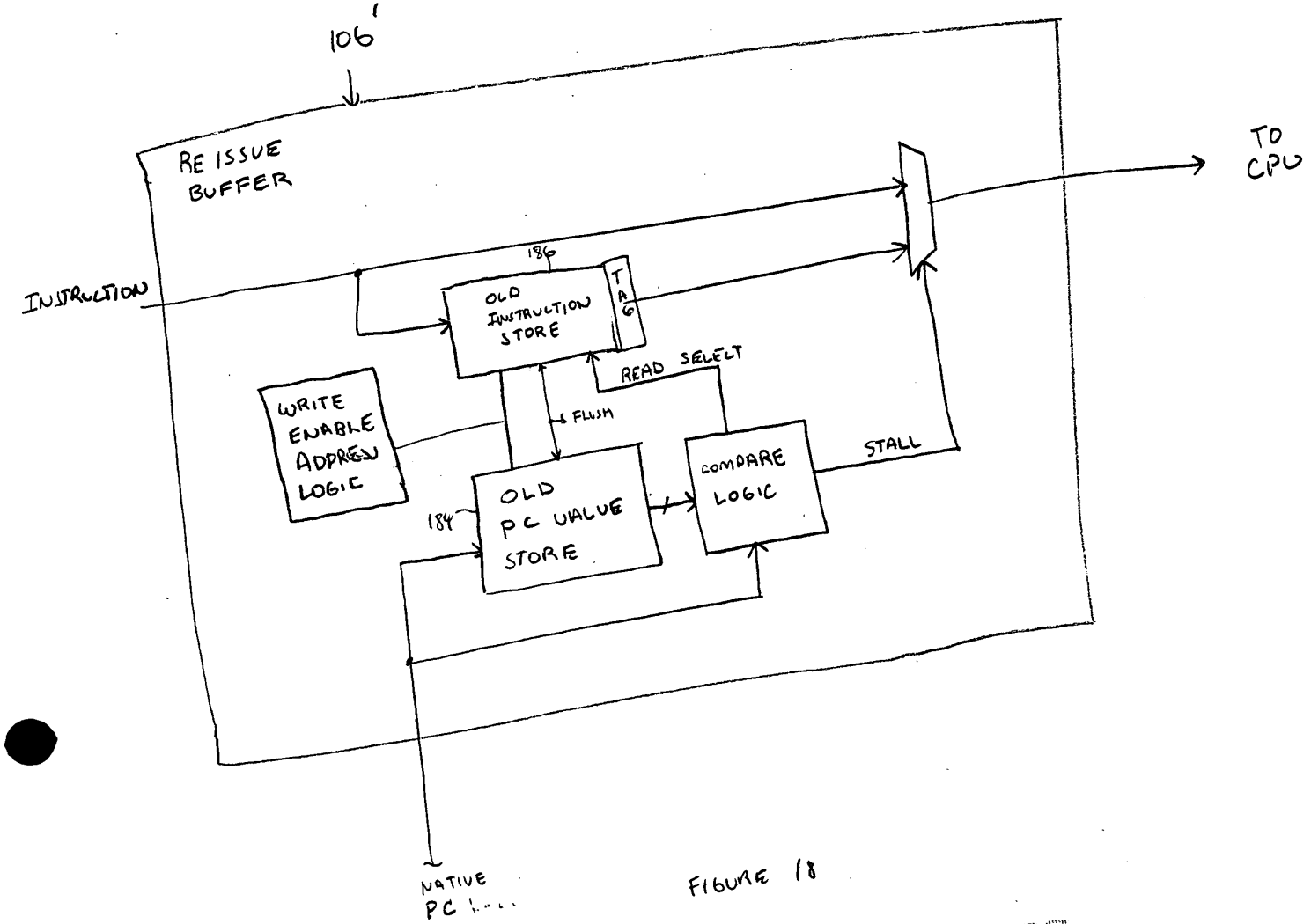

FIGURE 18

		TYPE		IDEAL COMBINATION	VARS TEST
iload 31	→	LD	→	LD	NO
iload 5	→	LD	→	LD	
iadd	→	OP	→	OP	
istore 8	→	ST	→		

DO LOAD OF VAR 31 from MEMORY

load var base stored in Stack manage into temp reg ste R1

Load word R1 + 31 (x 4) put result into the top of the stack

FIGURE 19

		TYPE	IDEAL COMBINATION	VARJ - TeJF
BYTECODE A → iload_3	→	LD →	LD	YES
BYTECODE B → iload 5	→	LD →	LD	YES
BYTECODE C → iAdd	→	OP →	OP	N/A
BYTECODE D → iconst_Ø	→	CONST		

VARS_H = 3
VARS_L = 5
OP TYPE = iadd
VAR_H control = 01
VAR_L CONTROL = 01
TOS modification = 1+1-1 = 1
BYTECODES USED = 3

FIGURE 20